Alcom-FT Technical Report Series
ALCOMFT-TR-02-50

Buckets strike back: Improved Parallel Shortest-Paths

Ulrich Meyer*
Max-Planck-Institut für Informatik (MPII),
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany,
www.uli-meyer.de

Abstract

We study the average-case complexity of the parallel single-source shortest-path (SSSP) problem, assuming arbitrary directed graphs with n nodes, m edges, and independent random edge weights uniformly distributed in $[0, 1]$. We provide a new bucket-based parallel SSSP algorithm that runs in $T = \mathcal{O}(\log^2 n \cdot \min_i\{2^i \cdot \mathbf{E}[\mathcal{L}] + |V_i|\})$ average-case time using $\mathcal{O}(n + m + T)$ work on a PRAM where $\mathcal{L}$ denotes the maximum shortest-path weight and $|V_i|$ is the number of graph vertices with in-degree at least 2^i. All previous algorithms either required more time or more work. The minimum performance gain is a logarithmic factor improvement; on certain graph classes, accelerations by factors of more than $n^{0.4}$ can be achieved. The algorithm allows adaptation to distributed memory machines, too.

1 Introduction

The *single-source shortest-path problem* (SSSP) is a fundamental and well-studied combinatorial optimization problem with many practical and theoretical applications [1]. Let $G = (V, E)$ be a directed graph with $|V| = n$ nodes and $|E| = m$ edges, let s be a distinguished vertex of the graph, and c be a function assigning a nonnegative real-valued *weight* to each edge of G. The objective of the SSSP is to compute, for each vertex v reachable from s, the weight of a minimum-weight ("shortest distance") path from s to v, denoted by $\mathrm{dist}(s, v)$, abbreviated $\mathrm{dist}(v)$; the weight of a path is the sum of the weights of its edges.

Assuming independent random edge weights is a standard setting for the *average-case analysis* of graph algorithms; see [14] for many examples. The uniform edge weight distribution is mostly chosen in order to keep the proofs simple. Frequently, the obtained results also hold asymptotically in the more general situation of random edge weights that are independent, bounded, and their common distribution function F has finite, nonnegative derivative at 0. This is also true for our algorithm.

The parallel random access machine (PRAM) [19] is one of the most widely studied abstract models of a parallel computer. A PRAM consists of P independent processors (processing units, PUs) and a shared memory, which these processors can synchronously access in unit time. The strongest model (CRCW) supports concurrent read- and write-access to single memory cells. The performance of PRAM algorithms is usually described by the two parameters *time* (assuming an unlimited number of available PUs) and *work* (the total number of operations needed). Even though the strict PRAM model is only implemented on a number of experimental parallel machines like the SB-PRAM [13], it is valuable to highlight the main ideas of a parallel algorithm without tedious details caused by a particular architecture. Other models like BSP [30] view a parallel computer as a collection of sequential processors, each one having its own local memory, so called *distributed memory machines (DMMs)*. The PUs are interconnected by a network that allows them to communicate by sending and receiving messages. In order to facilitate easy exposition we focus on the PRAM model and only sketch how our SSSP algorithm can be converted to DMMs.

1.1 Previous Work.

The classical *sequential* SSSP result is Dijkstra's algorithm [11]; implemented with Fibonacci heaps it solves SSSP on arbitrary directed graphs with nonnegative edge weights in $\mathcal{O}(n \cdot \log n + m)$ time. A number of faster algorithms have been developed on the more powerful RAM (random access machine) model, see [29] for an overview. In particular, Thorup [29] has given the first $\mathcal{O}(n + m)$ worst-case time RAM algorithm for *undirected* graphs with integer or float edge weights. The average-case analysis of shortest-path algorithms mainly focused on the *All–Pairs Shortest Paths* (APSP) problem for the *complete* graph with random edge weights. Recently, the first linear $\mathcal{O}(n + m)$ average-case time algorithms for *arbitrary directed* graphs with random edge weights have been given [16, 24].

So far there is no *parallel* $\mathcal{O}(n \cdot \log n + m)$ work PRAM SSSP algorithm with worst-case sublinear running time for arbitrary digraphs with nonnegative edge weights. The $\mathcal{O}(n \cdot \log n + m)$ work solution by Driscoll et. al. [12] has

*Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT), the DFG grant SA 933/1-1, and the Center of Excellence programme of the EU under contract number ICAI-CT-2000-70025. Parts of this work were done while the author was visiting the Computer and Automation Research Institute of the Hungarian Academy of Sciences, Center of Excellence, MTA SZTAKI, Budapest.

running time $\mathcal{O}(n \cdot \log n)$. An $\mathcal{O}(n)$ time algorithm requiring $\mathcal{O}(m \cdot \log n)$ work was presented by Brodal et. al. [7]. All faster known algorithms require more work, e.g., the approach by Han et. al. [18] needs $\mathcal{O}(\log^2 n)$ time and $\mathcal{O}(n^3 \cdot (\log\log n/\log n)^{1/3})$ work. The algorithm of Klein and Subramanian [20] takes $\mathcal{O}(\sqrt{n} \cdot \log \mathcal{L} \cdot \log n \cdot \log^* n)$ time and $\mathcal{O}(\sqrt{n} \cdot m \cdot \log \mathcal{L} \cdot \log n)$ work where $\mathcal{L}$ is the maximum shortest-path weight, i.e. $\mathcal{L} = \max_{v \in G, \mathrm{dist}(v) < \infty} \mathrm{dist}(v)$. Similar results have been obtained by Cohen [9] and Shi and Spencer [28]. Most of these algorithms can be modified to run on the weakest PRAM model without concurrent read/write capability.

Parallel shortest path problems on *random graphs* [6] where each of the n^2 possible edges is present with a certain probability have been studied intensively [8, 10, 15, 17, 25, 26, 27]. Under the assumption of independent random edge weights uniformly distributed in the interval $[0, 1]$ the fastest work-efficient parallel SSSP algorithm for random graphs [25, 26] requires $\mathcal{O}(\log^2 n)$ time and linear work on average; additionally, $\mathcal{O}(d \cdot \mathbf{E}[\mathcal{L}] \cdot \log n + \log^2 n)$ time and $\mathcal{O}(n + m + d \cdot \mathbf{E}[\mathcal{L}] \cdot \log n)$ work on average is sufficient for *arbitrary directed graphs* with random edge weights where d denotes the maximum node degree in the graph and $\mathcal{L}$ is defined as above.

For arbitrary graphs with large maximum degree d, the algorithms of [25, 26] perform poorly: $\Omega(d)$ time is needed. If the number of high-degree nodes is rather small, then the running time can be considerably improved [23]: let $|V_i|$ denote the number of graph vertices with in-degree at least 2^i then SSSP can be solved in $T_H = \mathcal{O}(\log^3 n \cdot \min_i\{2^i \cdot \mathbf{E}[\mathcal{L}] + |V_i|\})$ time on average. However, the algorithm needs non-linear $\mathcal{O}(n \cdot \log n + m + T_H)$ operations.

1.2 New Result.

We provide an improved parallel SSSP algorithm that applies different step-widths on disjoint node subsets at the same time and utilizes a new split-free bucket data structure. It achieves average-case running time $T = \mathcal{O}(\log^2 n \cdot \min_i\{2^i \cdot \mathbf{E}[\mathcal{L}] + |V_i|\})$ using $\mathcal{O}(n + m + T)$ operations on a CRCW PRAM, where $\mathcal{L}$ and $|V_i|$ are defined as in Section 1.1.

For sparse graphs, this means a logarithmic factor improvement on both running time and work bound compared to the superlinear work algorithm from [23]. Furthermore, the node degrees of many huge but sparse graphs with small diameter (e.g., WWW, telephone call graphs) follow a power law, i.e., the number of nodes with in-degree x is proportional to $x^{-\beta}$ for some constant β typically ranging between 2 and 4. For these graphs, the new approach is faster by a factor of nearly $n^{1/(\beta^2-\beta)}$ as compared to the best previous algorithm with linear average-case work ([25, 26]).

The rest of the paper is organized as follows: in Section 2, we shortly review basic facts and techniques for average-case efficient parallel and sequential shortest-path algorithms. Then, in Section 3, we present our new parallel SSSP algorithm. Finally, in Section 4, we give some concluding remarks.

2 Preliminaries

The classical sequential SSSP approach is Dijkstra's algorithm [11]. It maintains a partition of the node set V into *settled*, *queued*, and *unreached* nodes. For each node v it keeps a *tentative distance* $\mathrm{tent}(v)$. If v is unreached, then $\mathrm{tent}(v) = \infty$; otherwise, $\mathrm{tent}(v)$ refers to the weight of the lightest path from s to v found so far. Hence, $\mathrm{tent}(v) \geq \mathrm{dist}(v)$. Settled nodes satisfy $\mathrm{tent}(v) = \mathrm{dist}(v)$. Initially, s is queued ($s \in Q$), $\mathrm{tent}(s) = 0$, and all other nodes are unreached. In each iteration, the queued node v with smallest tentative distance is *scanned*: v is removed from the queue, and all edges (v, w) are *relaxed*, i.e., $\mathrm{tent}(w)$ is set to $\min\{\mathrm{tent}(w), \mathrm{tent}(v) + c(v, w)\}$. If w was unreached ($w \notin Q$), it is now queued. It is well known that $\mathrm{tent}(v) = \mathrm{dist}(v)$, when v is selected from the queue as the node with smallest tentative distance. Hence, v is settled and will never re-enter the queue. Therefore, Dijkstra's approach is a so called *label-setting* method. Alternatively, *label-correcting* variants may scan nodes from the queue for which $\mathrm{tent}(v) > \mathrm{dist}(v)$ and hence have to *rescan* those nodes until they are finally settled.

Label-correcting SSSP algorithms are natural candidates for parallelization: several queued nodes may be scanned concurrently in one round. However, finding both provably good criteria to select nodes for scanning and data structures that efficiently support these strategies remains a difficult task.

2.1 Buckets of Fixed Width.

The sequential SSSP algorithm of [25], called Δ-stepping, and its parallelizations [25, 26] are label-correcting approaches that work in phases: if M denotes the smallest tentative distance in the queue data structure Q at the beginning of a phase, then they scan queued nodes v with tentative distance $\mathrm{tent}(v) < M + \Delta$ in parallel. The parameter Δ is called the *step-width*. The queue Q is implemented by a linear array $B[\cdot]$ of buckets such that a queued node v is kept in $B[i]$ for tentative distances in the range $[i \cdot \Delta, (i+1) \cdot \Delta)$. Let k denote the largest bucket index such that $B[0], \ldots, B[k-1]$ are empty. Then a phase scans all nodes from the *current* bucket $B_{\text{cur}} = B[k]$ in parallel but first of all only relaxes the *light* edges (having weight at most Δ) emanating from these nodes. Any node v that is scanned from B_{cur} while $\mathrm{tent}(v) > \mathrm{dist}(v)$ is eventually reinserted into B_{cur}. Non-light edges out of v are only relaxed after B_{cur} finally remains empty. By then, v is surely settled; its non-light edges are relaxed using the final distance value for v. When B_{cur} becomes empty after a phase, then the algorithm sequentially searches for the next nonempty current bucket. Testing k buckets takes $\Theta(k + \log n)$ parallel time; thus, for maximum shortest-path weight $\mathcal{L}$, at least $\lceil \mathcal{L}/\Delta \rceil$ time is required. In order to obtain a reasonable parallel time bound, Δ should not be chosen too small. On the other hand, Δ must not be taken too large in order to avoid a high work bound due to numerous node rescans.

Lemma 1 ([25]) *For arbitrary directed graphs and independent random edge weights uniformly drawn from* $[0, 1]$*, the average-case number of rescans for an arbitrary node in the* Δ*-stepping algorithm is bounded by* $\mathcal{O}(1)$ *provided that* $\Delta \leq 1/d$ *for maximum node in-degree* d*.*

2.2 Adaptive Bucket-Splitting.

Based on the Δ-stepping algorithm sketched above, a sequential SSSP algorithm, **ADAP-SP**, with average-case running time $\mathcal{O}(n + m)$ was developed [24]. Starting with buckets of width $\Delta = 1$, it builds a bucket hierarchy in the following way: before the algorithm scans all nodes from the current bucket B_{cur} of width Δ_{cur}, it checks the maximum node degree d^* in B_{cur}. If $\Delta_{\text{cur}} > 1/d^*$ then it splits B_{cur} into smaller buckets of size $2^{-\lceil \log_2 d^* \rceil}$ each, and continues with the leftmost nonempty bucket among those buckets that were just generated.

Thus, on the one hand, the average-case number of reinsertions and re-relaxations can be bounded by $\mathcal{O}(n+m)$ since nodes with high degree are exclusively scanned from buckets with sufficiently small widths; on the other hand, the number of buckets is bounded by $\mathcal{O}(n + m)$ as well, independent of the maximum shortest-path weight $\mathcal{L}$.

2.3 Parallelization: Buckets versus Heaps.

A simple parallelization of **ADAP-SP** performs the operations for all nodes of the current bucket in parallel. Unfortunately, once a node with large degree d forces a reduction of the step-width in order to limit the risk of rescanning nodes, this step-width is kept for a certain distance range – even if no high-degree nodes remain in the new buckets of this range. As a consequence, $\Omega(d)$ time is spent to traverse these buckets. The drawback of excessive bucket traversals was partially removed in the *Parallel Degree Heap SSSP* algorithm [23], **PDH-SP** for short. **PDH-SP** uses a number of sequential priority queues and a simple method to compute an appropriate current step-width. As opposed to **ADAP-SP**, changing the step-width does not require restructuring of the priority queues themselves. However, this is paid for by a superlinear work bound.

A *sequential Degree Heap* D is a collection of $h = \lceil \log_2 n \rceil$ *relaxed heaps* $D_1, \ldots, D_h$ such that D_i is in charge of tentative distances for nodes having in-degree in $\{2^{i-1}, \ldots, 2^i - 1\}$. A relaxed heap allows insertions and decrease_key operations in worst-case constant time, deletions of the minimum in worst-case logarithmic time [12]. Let M_i be the smallest tentative distance in D_i ($M_i = \infty$ for empty D_i), and let $M = \min_i M_i$. Then the algorithm computes $\Delta_{\text{max}} := \min_i\{\max\{2^{-i-1}, M_i - M\}\}$. Subsequently, for each D_i, the SSSP algorithm scans all nodes $v \in D_i$ satisfying $\mathrm{tent}(v) < M + \Delta_{\text{max}}$. Hence, nodes with in-degree d are scanned using step-width at most $2^{-\lfloor \log_2 d \rfloor - 2}$ in order to limit the number of reinsertions. But whenever there are either no high-degree nodes left in the queue or they have sufficiently large tentative distance as compared to M, then the step-width is immediately increased to some larger value.

Parallel Degree Heaps are obtained by having a sequential Degree Heap D^j for each processor PU_j. A random mapping π is used to distribute the nodes over the sequential Degree Heaps. The global step-width for a phase can be found in poly-logarithmic time; no restructuring of the priority queues is required. Unfortunately, **PDH-SP** requires a superlinear number of operations due to the usage of heaps in order to decide which nodes to scan under changing global step-widths.

3 Parallel Individual Step-Widths

In this section we introduce the new parallel SSSP algorithm called **PIS-SP** (for **P**arallel **I**ndividual **S**tep-Widths). Whereas the old **PDH-SP** approach always uses a common step-width, **PIS-SP** applies *different* step-widths on disjoint node sets *at the same time* using a new split-free bucket data structure. Thus, the new approach is not just a re-implementation of **PDH-SP** with a specialized bucket structure but also utilizes a conceptually different node selection.

3.1 The Algorithm.

PIS-SP applies a queue data structure Q that consists of $\mathcal{O}(n)$ buckets in total, organized into $\mathcal{O}(\log n)$ arrays $B_i[\cdot]$: each array covers a total tentative distance range of width two. The buckets of each array are used in a cyclical fashion in order to subsequently store nodes with larger and larger tentative distances. The array $B_i[\cdot]$, $1 \leq i \leq \lceil \log_2 n \rceil =: h$ consists of 2^{i+2} buckets of width $\Delta_i := 2^{-i-1}$ each. $B_i[\cdot]$ exclusively stores nodes with in-degree in $\{2^{i-1}, \ldots, 2^i - 1\}$. Hence, for example a node v with in-degree $d = 6$ and tentative distance $\mathrm{tent}(v) = 5.3$ is stored in the array $B_i[\cdot]$, $i = \lfloor \log_2 d \rfloor + 1 = 3$, and more concretely in bucket $B_3[\lfloor (5.3 \bmod 2)/\Delta_3 \rfloor] = B_3[\lfloor 1.3/0.0625 \rfloor] = B_3[20]$. Note that there is no array for vertices with in-degree zero, as they cannot be reached anyway.

PIS-SP works as follows: initially, all nodes v reachable from the source node s via an edge (s, v) are put in parallel into their respective bucket arrays using the edge weights $c(s, v)$ as tentative distances. After that, **PIS-SP** operates in phases: At the beginning of a phase it first determines the globally smallest tentative distance $M := \min_{v \in Q} \mathrm{tent}(v)$ among all currently queued nodes. This step is quite intricate since the bucket structure only implements an approximate priority queue. Hence, M is not readily available; not too many operations must be used to find it. In Section 3.3 we will show how M can be obtained efficiently. Knowing M, **PIS-SP** scans all nodes from $B_i\left[\lfloor (M \bmod 2)/\Delta_i \rfloor\right]$ in parallel for each i, $1 \leq i \leq h$. This may insert new nodes into Q or reinsert previously scanned nodes with improved tentative distances. However, each phase settles at least one node. The algorithm stops if there are no queued nodes left after a phase. Hence, it requires at most n phases.

Each bucket is implemented by a separate array with dynamic space adjustment; additional arrays are used in order to keep track of tentative distances and to remember in which bucket and which array cell a queued node is currently

PIS-SP

foreach $(s, v) \in G$ **dopar**
  $i := \lfloor \log_2 \text{indegree}(v) \rfloor + 1$
  Insert v into $B_i[\lfloor c(s,v)/2^{-i-1} \rfloor]$
while $Q \neq \emptyset$ **do**
  Determine $M := \min_{v \in Q} \text{tent}(v)$ /* Phase starts */
  foreach $1 \leq i \leq \lceil \log_2 n \rceil$ **dopar**
    Empty bucket $B_i[\lfloor (M \bmod 2)/2^{-i-1} \rfloor]$
  $\Rightarrow$ node set R
  Build set *Req* for all edges out of R.
  Group *Req* by target nodes.
  foreach target node v of *Req* **dopar**
    Select best relaxation of an edge into v.
  Group selected edges by target buckets.
  Perform relaxations
  (& enlarge buckets if needed). /* Phase ends */

Figure 1. Pseudo-code for PIS-SP.

stored. Removing nodes from buckets in parallel is clearly congestion-free. However, insertions must be organized more carefully, see Figure 1: Having removed all nodes from the buckets $B_i[\lfloor (M \bmod 2)/2^{-i-1} \rfloor]$ for a phase (resulting in a node set R), the set *Req* of all edges emanating from nodes in R is built. An immediate parallel relaxation of the set *Req* might cause conflicts, therefore *Req* is first grouped by target nodes (using semi-sorting with small hashed values [5, 26]), and then the strictest relaxation request for each target node (group) is selected. The selected requests are grouped once more by target buckets and finally each group is appended in parallel after the last used position in the array for the target bucket. If there are not sufficiently many contiguous free positions left (the free positions may be scattered due to nodes that have been moved to other buckets during edge relaxations), then the whole content of this bucket is compacted and then copied to a new array of twice the size.

Each phase can be performed in $\mathcal{O}(\log n)$ average-case time; the respective number of operations excluding array size adjustments is linear in $|R| + |Req|$. The total work needed to adjust the array sizes can be amortized over the total number of edge (re-) relaxations.

3.2 Performance for Random Edge Weights.

In the following we consider the expected numbers of node rescans and phases for **PIS-SP** on graphs with random edge weights:

Definition 1 *Let $P := \langle v_j, \ldots, v_0 \rangle$ be a path into an arbitrary node v_0. P is called* **degree-weight balanced (dwb)** *if $c(v_{i+1}, v_i) \leq 2^{-\lfloor \log_2 \text{indegree}(v_i) \rfloor - 2}$, for all i, $0 \leq i < j$.*

Lemma 2 *For each node $v \in V$, the number of rescans during the execution of* **PIS-SP** *is bounded by the number of simple* **dwb** *paths into v.*

Proof: Let $\text{tent}^j(v)$ and Q_j denote the value of $\text{tent}(v)$ and the set of nodes in Q at the beginning of phase j, respectively; define $M^j := \min_{v \in Q_j} \text{tent}^j(v)$. Clearly, for nonnegative edge weights, $M^{j+1} \geq M^j$ and $\text{tent}^{j+1}(v) \leq \text{tent}^j(v)$ for all $j \geq 1$ and $v \in V$. Let $T(v,j)$ denote the total number of rescans for node v during the phases 1 to j; let $R(v,k,t)$ denote be the k-th rescan of he node v happening in phase t.

The proof of the lemma is by induction; we show that all rescans of the phases 1 to t can be injectively mapped onto simple paths. More specifically, a rescan of a node $v \in V$ in phase t is mapped onto a simple **dwb** path $P = \langle v_l, \ldots, v_1, v_0 = v \rangle$ of $l \leq t-1$ edges into v where v_j is scanned in phase $t-j$, and $\text{tent}^{t-j-1}(v_{j+1}) + c(v_{j+1}, v_j) = \text{tent}^{t-j}(v_j)$. It follows immediately that rescans of different nodes are mapped onto different simple paths. In the remainder we are concerned with different rescans of the same node.

Each node is scanned at most once per phase; no node is rescanned in the first phase. If the node v with degree d_v is scanned for the first time in phase j then it is scanned from bucket $B_{i_v}[\lfloor (M^j \bmod 2)/\Delta_{i_v} \rfloor]$ where $i_v = \lfloor \log_2 d_v \rfloor + 1$, and

$$\begin{aligned} M^j &\leq \text{dist}(v) \leq \text{tent}^j(v) < \lfloor M^j/\Delta_{i_v} \rfloor \cdot \Delta_{i_v} + \Delta_{i_v} \\ &\leq M^j + \Delta_{i_v} = M^j + 2^{-\lfloor \log_2 d_v \rfloor - 2} . \end{aligned}$$

Now we consider $R(v,k,t)$ where $t \geq k+1 \geq 2$ (i.e., $\text{tent}^{t-1}(v) > \text{tent}^t(v)$ and $T(v,t-1) = k-1$). As v was scanned for the first time in some phase $t' < t$, all nodes with final distances less than $M^{t'}$ were already settled at that time; their respective edges into v_0 (if any) will neither be relaxed in phase t' nor later. Therefore, $R(v,k,t)$ requires that some node $v' \neq v$ having $(v', v) \in E$ is scanned in phase $t-1$ (i.e., $\text{dist}(v') \geq M^{t-1} \geq M^{t'}$) where $\text{tent}^t(v) =$

$$\text{tent}^{t-1}(v') + c(v',v) < \text{tent}^{t-1}(v) < M^{t'} + 2^{-\lfloor \log_2 d_v \rfloor - 2}.$$

These conditions imply $c(v', v) \leq 2^{-\lfloor \log_2 d_v \rfloor - 2}$.

If $T(v', t-1) = 0$ then $R(v,k,t)$ can be uniquely mapped onto the edge (v', v), which is relaxed for the first time in phase $t-1$. Obviously, $\langle v', v \rangle$ is a simple **dwb** path into v of at most $t-1$ edges because $c(v', v) \leq 2^{-\lfloor \log_2 d_v \rfloor - 2}$.

Otherwise, i.e., if $T(v', t-1) = k' > 0$ then $R(v', k', t-1)$ was inductively mapped onto some simple **dwb** path $P' = \langle v'_{l'}, \ldots, v'_1, v'_0 = v' \rangle$ of $l' \leq t-2$ edges where v'_j is scanned in phase $t-1-j$, and $\text{tent}^{t-2-j}(v'_{j+1}) + c(v'_{j+1}, v'_j) = \text{tent}^{t-1-j}(v'_j)$. Hence, $\text{tent}^{t-1-j}(v'_j) \leq \text{tent}^{t-1}(v'_0)$, whereas $\text{tent}^{t-1-j}(v) \geq \text{tent}^{t-1}(v) > \text{tent}^{t-1}(v'_0)$ for all $0 \leq j < l'$. Consequently, v is not part of P'. We map $R(v,k,t)$ onto the path $P = \langle P', v \rangle$ that is built from the concatenation of P' and (v', v). As required, P is a simple **dwb** path of at most $t-1$ edges where the nodes are scanned in proper order and the equations for the tentative distances hold. Furthermore, P is different from any other path $\tilde{P} = \langle \tilde{P}', v \rangle$ constructed for some $R(v, \tilde{k}, \tilde{t})$ where $1 \leq \tilde{k} < \tilde{t} \leq k < t$: when constructing $\tilde{P}$ we either considered another edge $(v'', v) \neq (v', v)$, but then the subpaths P' and $\tilde{P}'$ end in different nodes; or we considered different rescans $R(v', k', t-1)$ and $R(v', k'', \tilde{t}-1)$ of the same node v', but then these rescans were mapped on different paths P' and $\tilde{P}'$ by induction. $\square$

Lemma 3 *For random edge weights uniformly drawn from* $[0, 1]$, **PIS-SP** *rescans each node at most* $\mathcal{O}(1)$ *times on the average.*

Proof: Let $X_l^{v_0}$ denote the number of simple **dwb** paths of $l \geq 1$ edges into an arbitrary node v_0 of a graph G. We first show $\mathrm{E}[X_l^{v_0}] \leq 2^{-l}$. The argument is by induction: let d_i denote the in-degree of node v_i. Excluding self-loops there is a set S_1 of at most d_0 edges into node v_0, hence $\mathrm{E}[X_1^{v_0}] \leq d_0 \cdot 2^{-\lfloor \log_2 d_0 \rfloor - 2} \leq d_0 \cdot 2^{-\log_2 d_0 - 1} = 1/2$.

Now consider the set $\mathcal{P}_l = \{p_1, \ldots p_k\}$ of all simple paths with l edges into node v_0 in G. By assumption $\mathrm{E}[X_l^{v_0}] = \sum_{p_i \in \mathcal{P}_l} \mathrm{P}[\, p_i \text{ is } \textbf{dwb} \,] \leq 2^{-l}$.

For each $p_i = \langle v_i, \ldots, v_0 \rangle \in \mathcal{P}_l$ there are at most d_i edges into v_i so that the concatenation with p_i results in a simple path of $l+1$ edges into v_0. In particular, for each such path $\langle v_j, v_i, \ldots, v_0 \rangle$, the weight of the newly attached edge (v_j, v_i) is independent of the weights on $\langle v_i, \ldots, v_0 \rangle$. Therefore, $\mathrm{P}[\, \langle v_j, v_i, \ldots, v_0 \rangle \text{ is } \textbf{dwb} \,] \leq 2^{-\lfloor \log_2 d_i \rfloor - 2} \cdot \mathrm{P}[\, \langle v_i, \ldots, v_0 \rangle \text{ is } \textbf{dwb} \,]$. By linearity of expectation, $\mathrm{E}[X_{l+1}^{v_0}]$ is bounded from above by

$$\begin{aligned} & \sum_{p_i \in \mathcal{P}_l} d_i \cdot 2^{-\lfloor \log_2 d_i \rfloor - 2} \cdot \mathrm{P}[\, p_i = \langle v_i, \ldots, v_0 \rangle \text{ is } \textbf{dwb} \,] \\ \leq & \sum_{p_i \in \mathcal{P}_l} 1/2 \cdot \mathrm{P}[\, p_i \text{ is } \textbf{dwb} \,] \leq 1/2 \cdot \mathrm{E}[X_l^{v_0}] \leq 2^{-l-1}. \end{aligned}$$

By Lemma 2, v_0 is rescanned at most $\sum_{j \geq 1} X_j^{v_0}$ times, therefore the average-case number of rescans of v_0 is at most $\sum_{j \geq 1} \mathrm{E}[X_j^{v_0}] \leq \sum_{j \geq 1} 2^{-j} = 1$. □

Lemma 4 *For graphs with random edge weights uniformly drawn from* $[0, 1]$, **PIS-SP** *needs* $r = \mathcal{O}(\log n \cdot \min_i \{2^i \cdot \mathbf{E}[\mathcal{L}] + |V_i|\})$ *phases on the average where* $\mathcal{L}$ *denotes the maximum shortest-path weight and* $|V_i|$ *is the number of graph vertices with in-degree at least* 2^i.

Proof: Let x, $0 \leq x \leq \lceil \log_2 n \rceil - 1$, be some arbitrary integer. For the analysis, let us distinguish two kinds of phases for the algorithm: B_x*-phases* scan only nodes with in-degree less than 2^x whereas S_x*-phases* scan at least one node with in-degree 2^x or larger. Let s_x denote the number of S_x-phases. By Lemma 3, each node is rescanned $\mathcal{O}(1)$ times on average. Hence, $\mathbf{E}[s_x] = \mathcal{O}(|V_x|)$.

In the following we are interested in B_x*-chunks*: for some constant $c \geq 1$, a B_x-chunk consists of $t = (c+5) \cdot \log_2 n + 1$ consecutive B_x-phases without any intermediate S_x-phases. Observe that less than $(s_x + 1) \cdot t$ of all B_x-phases do not belong to any B_x-chunk. Let M^j be the smallest tentative distance among queued nodes at the beginning of the j-th phase of a B_x-chunk. We will show

$$\lfloor M^t / \Delta_x \rfloor > \lfloor M^1 / \Delta_x \rfloor \quad \text{with high probability}^1. \qquad (1)$$

[1] For a problem of size n, we say that an event occurs *with high probability* (whp) if it occurs with probability at least $1 - n^{-c}$ for an arbitrary but fixed constant $c \geq 1$.

During the j-th phase of the B_x-chunk, **PIS-SP** scans all nodes from the current buckets $B_i\left[\lfloor (M^j \bmod 2)/\Delta_i \rfloor\right]$, $1 \leq i \leq x$. Let $\mathcal{N}_j$ denote the set of these nodes. Observe that (1) holds if $\lfloor (M^1 \bmod 2)/\Delta_i \rfloor \neq \lfloor (M^t \bmod 2)/\Delta_i \rfloor$ for at least one i, $1 \leq i \leq x$, i.e., **PIS-SP** has advanced at least one current bucket from $B_1[\cdot], \ldots, B_x[\cdot]$ according to the cyclical ordering. So, let us assume that the current buckets from $B_1[\cdot], \ldots, B_x[\cdot]$ in phase t are those of phase 1. But then there must be at least one node $v_t \in \mathcal{N}_t$ with $\mathrm{tent}(v_t) < M^1 + \Delta_x$. In particular, there must be a simple **dwb** path $P = \langle v_1, \ldots v_t \rangle$ of total weight less than $\Delta_x = 2^{-x-1}$ where $v_j \in \mathcal{N}_j$, and all v_j, $1 \leq j \leq t$, have in-degree less than 2^x. Into any node $v \in \mathcal{N}_t$ there are less than $2^{x \cdot (t-1)}$ such paths. As shown in [25], the sum of l independent random edge weights (uniformly distributed in $[0, 1]$) is at most $\Delta \leq 1$ with probability at most $\Delta^l / l!$. Hence, the probability that such a path exists into any node $v \in \mathcal{N}_t$ is bounded by $n \cdot 2^{x \cdot (t-1)} \cdot \Delta_x^{t-1} / (t-1)! \leq n \cdot 2^{-t+1} \leq n^{-c-4}$. That proves (1). Therefore, after $\lceil \mathcal{L}/\Delta_x \rceil$ B_x-chunks the current buckets have been advanced so much that no node remains in the bucket structure with probability at least $1 - \lceil \mathcal{L}/\Delta_x \rceil \cdot n^{-c-4} \geq 1 - n^{-c-1}$. This accounts for at most another $\mathcal{O}(2^x \cdot \mathcal{L} \cdot \log n)$ B_x-phases with probability at least $1 - n^{-c-1}$; as **PIS-SP** requires at most n phases in the worst case, it needs at most $\mathcal{O}(2^x \cdot \mathbf{E}[\mathcal{L}] \cdot \log n + n \cdot n^{-c-1})$= $\mathcal{O}(2^x \cdot \mathbf{E}[\mathcal{L}] \cdot \log n)$ additional B_x-phases on the average.

Altogether the algorithm runs in $\mathcal{O}((2^x \cdot \mathbf{E}[\mathcal{L}] + |V_x|) \cdot \log n)$ phases on the average. Since this analysis holds for any integer x, the average-case bound for all phases can be restated as $\mathcal{O}(\min_i \{2^i \cdot \mathbf{E}[\mathcal{L}] + |V_i|\} \cdot \log n)$. Note once more that the algorithm itself does not have to find an optimal compromise between B_x-phases and S_x-phases; they are just a theoretical concept of the analysis. □

3.3 Fast Node Selection.

In the following we show how the bucket data structure can be modified in order to determine M fast and efficiently: each array $B_i[\cdot]$ is augmented by a pointer to the first nonempty bucket B_i^* of range $[M_i^*, M_i^* + \Delta_i]$ according to the cyclic ordering.

Lemma 5 *For random edge weights uniformly drawn from* $[0, 1]$, *the total number of operations needed to maintain the pointers to* B_i^* *during the execution of* **PIS-SP** *is bounded by* $\mathcal{O}(n + m)$ *on the average. For each phase, maintaining the pointers takes at most* $\mathcal{O}(\log n)$ *time.*

Proof: B_i^* may have changed after a phase due to (i) an insertion of a node v into $B_i[\cdot]$, (ii) a decrease of $\mathrm{tent}(v)$ where $v \in B_i[\cdot]$, (iii) a scan of v from $B_i[\cdot]$. Cases (i) and (ii) are straightforward: if v moves to a bucket dedicated to smaller tentative distances then B_i^* is set to this new bucket otherwise it stays unchanged. The grouping and selection procedure taking place in each phase prior to parallel node insertions can be adapted such that for each array the smallest inserted tentative distance is determined as well. Only this value needs to be checked for a possible update of B_i^*.

Maintaining B_i^* for case (iii) is only non-trivial if the bucket of the scanned node v remains empty after a phase: all buckets of $B_i[\cdot]$ must be checked in order to find the new first nonempty bucket according to the cyclic ordering. On a PRAM this can be accomplished using standard parallel prefix-sum and minimum computations [19] in $\mathcal{O}(\log n)$ time and $\mathcal{O}(2^{i+2})$ work. These operations can be attributed to the degree of the scanned node v, which is at least 2^{i-1}. Since all degrees add up to $\mathcal{O}(m)$ and since each node is scanned $\mathcal{O}(1)$ times on average (Lemma 3), the total amount of operations needed for all occurrences of case (iii) is bounded by $\mathcal{O}(n + m)$ on the average, as well. $\square$

Finding the Minimum. We show how the values M_i^* of the first nonempty buckets $B_i^* = [M_i^*, M_i^* + \Delta_i)$ introduced above are used to identify the value M of the globally smallest tentative distance among all queued nodes. At the beginning of a phase **PIS-SP** computes the *suffix-minima* of the values M_i^*, i.e., for each i, it builds $M(i) := \min\{M_i^*, M_{i+1}^*, \ldots, M_h^*\}$. We set $M_i^* := +\infty$ if $B_i[\cdot]$ is empty. Note that $M(1) \leq M < M(1) + \Delta_1$. Next, the procedure refines safe estimates $\widehat{M_i} \leq M \leq \widehat{T_i}$ for the globally smallest tentative distance among all queued nodes: initially, we set $\widehat{M_1} := M(1)$ and $\widehat{T_1} := M(1) + \Delta_1$. The minimum detection takes place in at most h stages:

For the first stage, **PIS-SP** checks the bucket $B_1\left[\left\lfloor(\widehat{M_1} \bmod 2)/\Delta_1\right\rfloor\right]$: either, this bucket is empty, or it constitutes the bucket B_1^* keeping the node with smallest tentative distance for the array $B_1[\cdot]$. All nodes from $B_1^* = B_1\left[\left\lfloor(\widehat{M_1} \bmod 2)/\Delta_1\right\rfloor\right]$ can be scanned during the phase according to the scan criterion. Let T_1 be the smallest tentative distance among all nodes of B_1^* ($T_1 := \infty$ if B_1^* is empty). The value of T_1 is determined with a parallel minimum algorithm causing work linear in the number of elements stored in bucket B_1^*. If $\min\{T_1, \widehat{T_1}\} \leq M(2)$ then either the whole bucket structure is empty or a node v' with globally smallest tentative distance is stored in B_1^*, and therefore $M = \min\{T_1, \widehat{T_1}\}$. The procedure stops. Otherwise, if $\min\{T_1, \widehat{T_1}\} > M(2)$, the queued node with smallest tentative distance may still be found in one of the subsequent arrays $B_2[\cdot], \ldots, B_h[\cdot]$. Therefore, the procedure sets $\widehat{M_2} := M(2)$, $\widehat{T_2} := \min\{T_1, \widehat{T_1}, M(2) + \Delta_2\}$, and it continues with stage 2 on the bucket array $B_2[\cdot]$.

In general, if the detection procedure reaches stage $i \geq 2$ then we may assume by induction that it has identified estimates $\widehat{M_i} \leq M < \widehat{T_i} \leq \widehat{M_i} + \Delta_i$. The procedure computes the smallest distance value T_i among the nodes in $B_i\left[\left\lfloor(\widehat{M_i} \bmod 2)/\Delta_i\right\rfloor\right]$. Observe that this bucket is either empty (then $T_i := +\infty$) or its nodes will be scanned by **PIS-SP**. Therefore, the work to identify T_i can be amortized over the node scans. Having computed T_i we may find $\min\{T_i, \widehat{T_i}\} \leq M(i+1)$, then $\min\{T_i, \widehat{T_i}\}$ definitely equals M; the procedure stops. Otherwise, that is if $\min\{T_i, \widehat{T_i}\} > M(i+1)$, we only know for sure that $M(i+1) \leq M_i \leq \min\{T_i, \widehat{T_i}, M(i+1) + \Delta_{i+1}\}$. The procedure will continue in $B_{i+1}[\cdot]$ with the new estimates: $\widehat{M_{i+1}} := M(i+1)$ and $\widehat{T_{i+1}} := \min\{T_i, \widehat{T_i}, M(i+1) + \Delta_{i+1}\}$. After at most $h = \mathcal{O}(\log n)$ stages, M is eventually determined. Figure 2 provides an example.

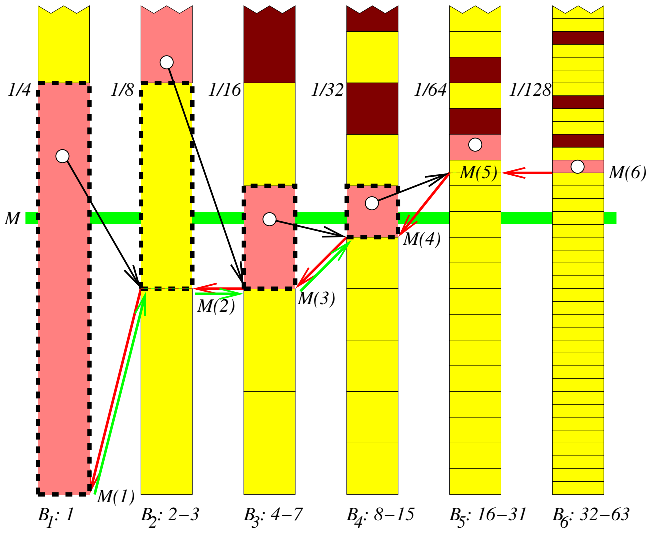

Figure 2. Determination of the smallest tentative distance M in the bucket structure. The circles denote the smallest distance T_i in each array $B_i[\cdot]$. In this example, only the nodes in the first nonempty buckets of $B_1[\cdot]$, $B_3[\cdot]$, and $B_4[\cdot]$ are examined to obtain M: after testing the nodes from the first nonempty bucket of $B_1[\cdot]$ we find $T_1 > M(2)$; hence, the bucket $B_2[\lfloor(M(2) \bmod 2)/\Delta_2\rfloor]$ is tested, but this bucket is empty. As $\widehat{T}_2\ (= T_1) > M(3)$, the search continues in $B_3[\lfloor(M(3) \bmod 2)/\Delta_3\rfloor]$ where we find a new smallest element (T_3), but it is still larger than $M(4)$. We continue in $B_4[\lfloor(M(4) \bmod 2)/\Delta_4\rfloor]$ where no smaller element is found. Finally, $M(5)$ is larger than the smallest distance seen so far (T_3). Thus, all other elements in $B_i[\cdot]$, $i \geq 5$ will be larger as well, therefore $M = T_3$.

Scanning the nodes from the examined buckets can be overlapped with the global minimum detection. However, asymptotically this does not improve the running time. The time required for a phase depends on the applied procedure for the $\mathcal{O}(\log n)$ local minimum computations: with the standard approach based on balanced trees [19] a phase of **PIS-SP** needs $\mathcal{O}(\log^2 n)$ time and $\mathcal{O}(k + \log^2 n)$ work where k denotes the number of nodes scanned in that phase. Using the constant-time linear-work randomized minimum computation from [15] reduces the time by a logarithmic factor.

Combining the results of this section we find:

Theorem 1 *For graphs with random edge weights uniformly drawn from* $[0, 1]$, **PIS-SP** *requires* $T = \mathcal{O}(\log^2 n \cdot \min_i\{2^i \cdot \mathcal{L} + |V_i|\})$ *average-case time using* $\mathcal{O}(n + m + T)$ *operations on a PRAM where* $\mathcal{L}$ *denotes the maximum shortest-path weight and* $|V_i|$ *is the number of graph vertices with in-degree at least* 2^i.

3.4 Conversion to Distributed Memory Machines.

The straightforward way to obtain a DMM algorithm out of a PRAM algorithm is to use an efficient PRAM simulation method: given certain conditions, a step of a CRCW PRAM with P'-processors can be simulated on a P-processors BSP machine in $\mathcal{O}(P'/P)$ time if $P' \geq P^{1+\epsilon}$ for any constant $\epsilon > 0$; see [30]. Hence, if T is defined as in Theorem 1, the distributed algorithm runs in $\mathcal{O}(P^\epsilon \cdot T + \frac{n+m+T}{P})$ time.

Alternatively, for a direct coding on a DMM machine, one first has to solve the data allocation problem: all arrays and long adjacency lists are spread over the local memories of the processors using random hash functions. If the number of processors, P, is reasonably bounded then the accesses to the memory modules are sufficiently load-balanced in each phase with high probability. Non-local memory access is realized via message passing and random routing. Each phase of the algorithm can be performed in a number of *supersteps* each of which consists of local computations, synchronization, and communication. Important ingredients for the conversion are standard DMM implementations for tree based reduction and broadcasting schemes [19]. They are needed for prefix-sum / minimum computations and distribution of values, e.g., when all edges (v, w_i) of a high-degree node v are to be relaxed, then the value of $\mathrm{tent}(v)$ must be made available to all PUs that store outgoing edges of v. Figure 3 depicts the role of spreading and grouping for load balancing. The grouping steps of the algorithm can be implemented by DMM integer-sorting algorithms, e.g., [3]. The choice of the sorting algorithm determines how many supersteps are needed to implement a phase of the PRAM algorithm, and hence how many processors can be reasonably used.

3.5 Performance Gain on Power Law Graphs.

Many sparse massive graphs such as the WWW graph and telephone call graphs share universal characteristics which can be described by the so-called "power law" [2, 4, 21]: the number of nodes, y, of a given in-degree x is proportional to $x^{-\beta}$ for some constant $\beta > 0$. For most massive graphs, $2 < \beta \leq 4$: independently, Kumar et. al. [21] and Barabasi et. al. [4] reported $\beta \approx 2.1$ for the in-degrees of the WWW graph, and the same value was estimated for telephone call graphs [2]. Further studies mentioned in [4] on social networks resulted in $\beta \approx 2.3$, and for graphs of the electric power grid in the US, $\beta \approx 4$.

The observed diameters are usually very small; on random graph classes – with $\mathcal{O}(n)$ edges – that are widely considered to be appropriate models of real massive graphs like

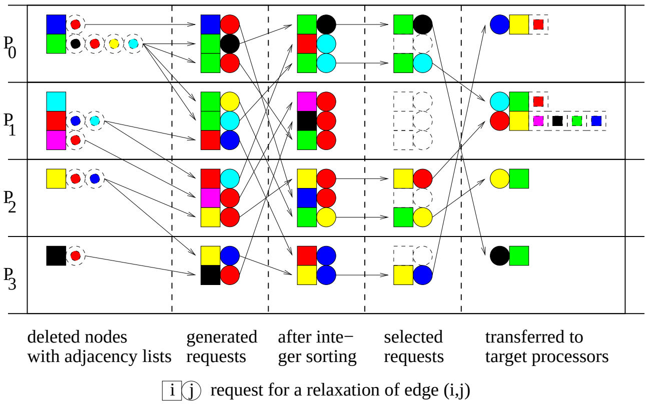

Figure 3. Generating and performing relaxation requests: requests are denoted by a box for the source node and a circle for the target node, colors are used to code node indices. The processors cooperate in building the total set of requests: large adjacency lists are handled by groups of PUs. Subsequently, the generated requests are grouped by target nodes using integer-sorting. Then superfluous requests are filtered out, and the remaining requests are sent to the processors which host the appropriate bucket structures. Without the spreading and grouping, processor P_0 would be overloaded during the generation, and processor P_1 would receive too many requests.

the WWW it turns out that $\mathcal{L} = \mathcal{O}(\log n)$ whp [22]. For such graphs the number of nodes having in-degree at least d^* is approximately given by $\Theta(n \cdot \sum_{x \geq d^*} x^{-\beta})$, which for constant $\beta \geq 2$ and arbitrary $d^* \geq 1$ can be bounded by $\Theta(n \cdot \int_{d^*}^{\infty} x^{-\beta}\, dx) = \Theta(n \cdot d^{*\,-\beta+1})$. Taking $d^* = n^{1/\beta}$ one expects $\mathcal{O}(n \cdot n^{1/\beta \cdot (-\beta+1)}) = \mathcal{O}(n^{1/\beta})$ nodes with in-degree at least d^*. On the other hand, we expect at least one node with in-degree $d = \Omega(n^{\frac{1}{\beta-1}})$ by solving $n \cdot d^{-\beta+1} = 1$.

Hence, assuming independent random edge weights uniformly distributed in $[0, 1]$, the average-case time for SSSP on WWW-like graphs can be estimated as follows: The previous parallel SSSP algorithms from [25, 26] that achieve linear average-case work on these graph classes require $\Omega(n^{\frac{1}{\beta-1}})$ time since they have to traverse $\Omega(d) = \Omega(n^{\frac{1}{\beta-1}})$ buckets sequentially. Our new approach requires $\mathcal{O}(\log^2 n \cdot (n^{1/\beta} \cdot \log n + n^{1/\beta})) = \mathcal{O}(n^{1/\beta} \cdot \log^3 n)$ average-case time using linear $\mathcal{O}(n+m)$ work. Hence, while retaining the linear work bound, the average-case running time could be improved by a factor of $\Omega(n^{1/(\beta^2-\beta)} / \log^3 n)$.

4 Conclusions

We have given a new parallel SSSP algorithm together with a powerful split-free bucket data-structure. It facilitates improved average-case time linear-work SSSP computations for many graph classes with small diameter.

In order to obtain fast SSSP algorithms with linear average-case work, one might also try to parallelize Goldberg's new sequential label-setting algorithm [16]. However, in its current form, the criterion used to detect nodes that can be scanned in arbitrary order is at most as efficient as the IN-approach in [10]. It can be shown that there are graph classes with random edge weights, maximum constant node degree and maximum shortest-path weight $\mathcal{L} = \mathcal{O}(\log n)$ where this criterion requires $\Omega(\sqrt{n}/\log n)$ phases. In contrast, our new approach runs in $\mathcal{O}(\log^3 n)$ time and linear work for these graphs. Still, there may be better label-setting criteria.

For the future it would be desirable to solve the SSSP on Web-like graphs in poly-logarithmic time using at most $\mathcal{O}(n \cdot \log n + m)$ work. Furthermore, any work-efficient algorithm with sublinear running time that is independent of the diameter would be of great interest.

Acknowledgements

The author would like to thank Annamária Kovács for giving valuable comments on a previous version of this paper.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proc. 32nd Annual ACM Symposium on Theory of Computing*, pages 171–180. ACM, 2000.

[3] D. A. Bader, D. R. Helman, and J. Jájá. Practical parallel algorithms for personalized communication and integer sorting. *Journal of Experimental Algorithms*, 3(1):1–42, 1996.

[4] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[5] H. Bast and T. Hagerup. Fast parallel space allocation, estimation and integer sorting. *Information and Computation*, 123:72 – 110, 1995.

[6] B. Bollobás. *Random Graphs*. Academic Press, 1985.

[7] G. S. Brodal, J. L. Träff, and C. D. Zaroliagis. A parallel priority queue with constant time operations. *Journal of Parallel and Distributed Computing*, 49(1):4–21, 1998.

[8] A. Clementi, J. Rolim, and E. Urland. Randomized parallel algorithms. In *Solving Combinatorial Problems in Parallel*, volume 1054 of *LNCS*, pages 25–50, 1996.

[9] E. Cohen. Using selective path-doubling for parallel shortest-path computations. *Journal of Algorithms*, 22(1):30–56, Jan. 1997.

[10] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of Dijkstra's shortest path algorithm. In *23rd Symp. on Mathematical Foundations of Computer Science*, volume 1450 of *LNCS*, pages 722–731. Springer, 1998.

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. *Num. Math.*, 1:269–271, 1959.

[12] J. R. Driscoll, H. N. Gabow, R. Shrairman, and R. E. Tarjan. Relaxed heaps: An alternative to fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31, 1988.

[13] A. Formella, J. Keller, and T. Walle. HPP: A high performance PRAM. In *Proc. Euro-Par 1996 Parallel Processing*, volume II of *LNCS 1124*, pages 425–434, Berlin, August 1996. Springer.

[14] A. Frieze and C. McDiarmid. Algorithmic theory of random graphs. *Random Structures and Algorithms*, 10:5–42, 1997.

[15] A. M. Frieze and L. Rudolph. A parallel algorithm for all-pairs shortest paths in a random graph. In *Proc. 22nd Allerton Conference on Communication, Control and Computing*, pages 663–670, 1985.

[16] A. V. Goldberg. A simple shortest path algorithm with linear average time. In *Proc. 9th Ann. European Symposium on Algorithms (ESA)*, number 2161 in LNCS, pages 230–241. Springer, 2001.

[17] Q. P. Gu and T. Takaoka. A sharper analysis of a parallel algorithm for the all pairs shortest path problem. *Parallel Computing*, 16(1):61–67, 1990.

[18] Y. Han, V. Pan, and J. Reif. Efficient parallel algorithms for computing all pair shortest paths in directed graphs. *Algorithmica*, 17(4):399–415, 1997.

[19] J. Jájá. *An Introduction to Parallel Algorithms*. Addison-Wesley, Reading, 1992.

[20] P. N. Klein and S. Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 25(2):205–220, Nov. 1997.

[21] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *Proc. 8th International World-Wide Web Conference*, 1999.

[22] L. Lu. The diameter of random massive graphs. In *Proc. 12th Ann. Symp. on Discrete Algorithms*, pages 912–921. ACM–SIAM, 2001.

[23] U. Meyer. Heaps are better than buckets: Parallel shortest paths on unbalanced graphs. In *Proc. Euro-Par 2001 Parallel Processing*, volume 2150 of *LNCS*, pages 343–351. Springer, 2001.

[24] U. Meyer. Single-source shortest-paths on arbitrary directed graphs in linear average-case time. In *Proc. 12th Ann. Symp. on Discrete Algorithms*, pages 797–806. ACM–SIAM, 2001.

[25] U. Meyer and P. Sanders. Δ-stepping: A parallel shortest path algorithm. In *Proc. 6th Ann. European Symposium on Algorithms (ESA)*, volume 1461 of *LNCS*, pages 393–404. Springer, 1998.

[26] U. Meyer and P. Sanders. Parallel shortest path for arbitrary graphs. In *Proc. Euro-Par 2000 Parallel Processing*, volume 1900 of *LNCS*, pages 461–470. Springer, 2000.

[27] J. Reif and P. Spirakis. Expected parallel time and sequential space complexity of graph and digraph problems. *Algorithmica*, 7:597–630, 1992.

[28] H. Shi and T. H. Spencer. Time–work tradeoffs of the single-source shortest paths problem. *Journal of Algorithms*, 30(1):19–32, 1999.

[29] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.

[30] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.